HPCML: A Modeling Language Dedicated to High-Performance Scientific Computing

Marc Palyart
CEA/CESTA
33114 Le Barp, France
marc.palyart@cea.fr

David Lugato
CEA/CESTA
33114 Le Barp, France
david.lugato@cea.fr

Ileana Ober
IRIT – Université de Toulouse
118, route de Narbonne
31062 Toulouse, France
ober@irit.fr

Jean-Michel Bruel
IRIT – Université de Toulouse
118, route de Narbonne
31062 Toulouse, France
bruel@irit.fr

ABSTRACT

Tremendous computational resources are required to compute complex physical simulations. Unfortunately computers able to provide such computational power are difficult to program, especially since the rise of heterogeneous hardware architectures. This makes it particularly challenging to exploit efficiently and sustainably supercomputers resources. We think that model-driven engineering can help us tame the complexity of high-performance scientific computing software development by separating the different concerns such as mathematics, parallelism, or validation. The principles of our approach, named MDE4HPC, stem from this idea. In this paper, we describe the High-Performance Computing Modeling Language (HPCML), a domain-specific modeling language at the center of this approach.

Categories and Subject Descriptors

D.1.3 [**Programming Techniques**]: Concurrent Programming—*Parallel programming*; D.1.7 [**Programming Techniques**]: Visual Programming; D.2.2 [**Software Engineering**]: Design Tools and Techniques; D.3.2 [**Programming Languages**]: Language Classifications—*Specialized application languages; Concurrent, distributed, and parallel languages; Design languages; Very high-level languages*

General Terms

Design; Language; Performance

Keywords

Model-Driven Engineering; Domain-Specific Modeling Language; Scientific Computing; High-Performance Computing; Parallel Language

1. INTRODUCTION

While software industry is facing the multicore crisis, high-performance scientific computing developers had to initiate the shift to parallel programming a long time ago. Indeed parallel architectures were the only way to quench their thirst for computational power.

Unfortunately, mainstream parallel programming models, and in particular those addressing high-performance computing (HPC), are generally low level and machine specific. Even though good performance levels can be achieved with these approaches, drawbacks in terms of programming complexity, architecture dependency and mix-up of concerns occur :

- The complexity of parallel software programming restricts the use of these workstations and supercomputers to a few scientists (usually they are not computer scientists) who are willing to spend a significant amount of time learning the specificities of a particular set of machines.
- In our case, the life cycle of supercomputers is five to seven times shorter than the life cycle of scientific applications that run on them (4 years versus 20 to 30 years). In addition to usual problems encountered with software maintenance over such a period of time (e.g. team turnover) software migrations have to face radical changes in hardware architectures arisen from the race for performance.
- The problem to be solved - the scientific knowledge of the physics - is entirely mixed with numerical schemes and target dependent information added to manage the parallelism. Once a complex system has been built, it is difficult to identify how the problem is solved by the the application. As a result, maintenance and upgrading become even more complicated.

We think that model-based development techniques can help us deal with these problems. First by offering abstract and domain-specific concepts to developers (mainly applied mathematics researchers or physicists) for specifying how

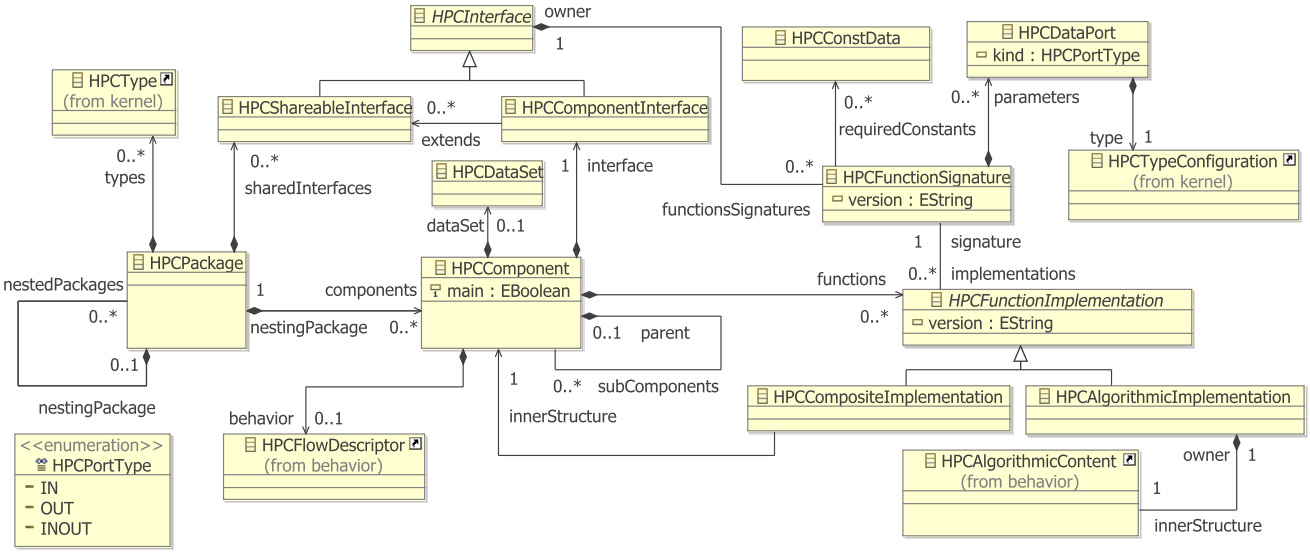

Figure 1: HPCML Structure Package

to solve their problems. Second by reducing the cost and complexity of the software migration process thanks to the sharing between projects of model transformations used to convert abstract representations into executable implementations. Finally by separating the concerns with different models and views located at different levels of abstraction.

In accordance with this opinion, we described in [15] the characteristics and possibilities of such a development approach that we called MDE4HPC. In [14], we applied the MDE4HPC principles with the ArchiMDE tool and attempted to combine MDE and frameworks in order to solve our problem. Based on the encouraging results of this last experiment regarding possible reduction of maintenance costs and thanks to the positive feedback from applied mathematics researcher on the accessibility of the approach we improved MDE4HPC and are evaluating it on a larger scale application development.

In this paper we carry on the presentation of the MDE4HPC approach and focus our attention on the High-Performance Computing Modeling Language (HPCML), its core component. The principal contribution of this paper is the presentation of the concepts offered by the new version of HPCML to model parallel applications independently from a specific platform. We believe that our metamodel partially presented in [14] was to tied to the selected framework, thus the version of HPCML presented here attempts to be more generic.

The rest of this paper is organised as follows: in Section 2 we present the structure package of the HPCML metamodel and its component model. In Section 3 we follow with the behavior package that provides constructs for expressing parallelism. After presenting related work in Section 4, we finally discuss in Section 5 the contributions of our research and give directions for future work.

2. STRUCTURAL ASPECT

Numerical simulation programs are functional by nature, they take input data, process them and give a result. The Fortran language designed by John Backus in the late 1950s for the IBM 704 a precursor of supercomputers, is still the reference of the scientific computing community. It is a procedural languages that decompose a program into a hierarchy of functions. One other strength of Fortran comes from its capacity to manipulate natively mathematical data types such as matrix or vector. The metamodel of the structure package presented in Figure 1 arises from this need for functional decomposition.

The main concept of the structure package is the *HPCComponent*. The service provided by an *HPCComponent* is defined with an *HPCComponentInterface* that may conform to interfaces shared by several components (*HPCShareableInterface*).

The implementation of the component interface is delegated to an *HPCFunctionImplementation*. A particularity of these functions is that they can either be called from another components that requests the service or from the component itself if it possesses an execution semantic defined with an *HPCFlowDescriptor* (see section 3). Excluding constants, an *HPCComponent* does not hold data that could be shared among its internal functions. These functions therefore declare all the data they consume and produce (*HPCDataPort* and *HPCConstData*). This is a deliberate choice, dictated by the need to identify data dependencies that are truly required.

Two possibilities are offered to the developer for defining an *HPCFunctionImplementation*, he can:

- decide to refine his model with an *HPCCompositeImplementation*. In this case he needs to model a subcomponent with its associated *HPCFlowDescriptor*.

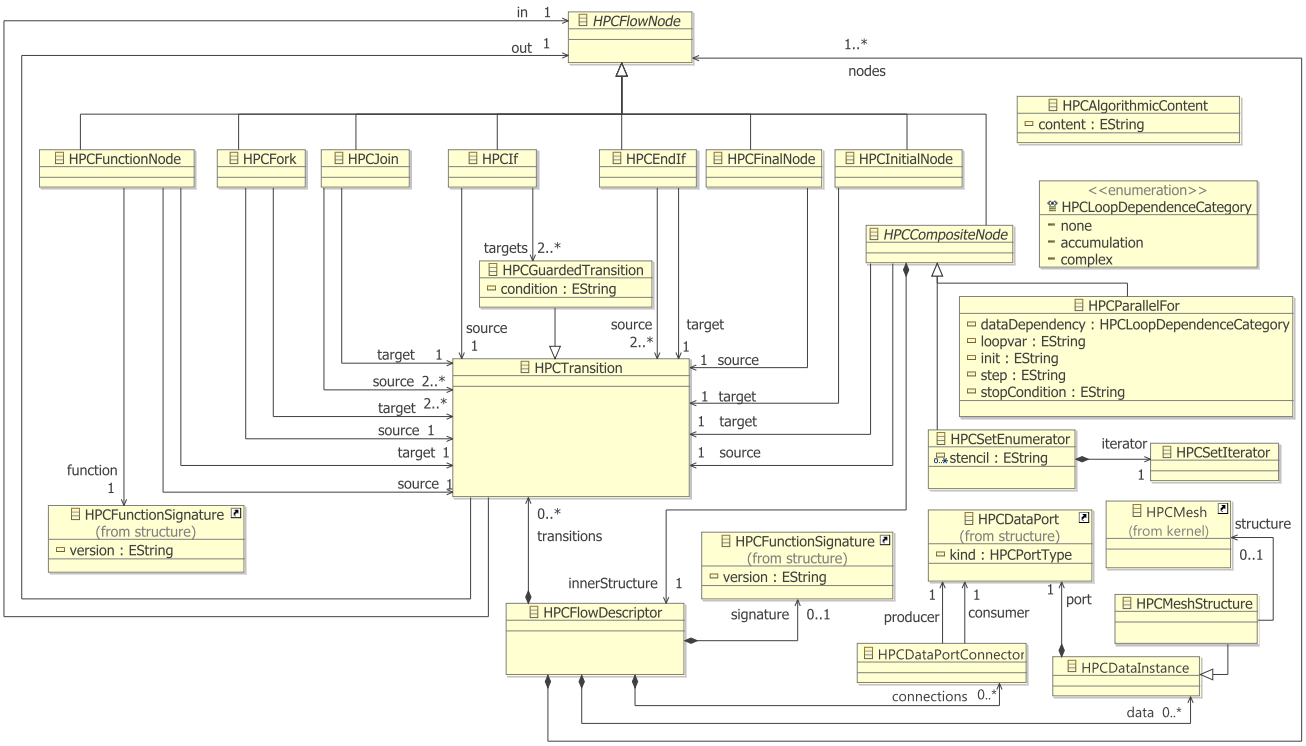

Figure 2: HPCML Behavior Package

- consider that the function is simple enough to be described with a textual language (*HPCAlgorithmicContent*). Even though MDE4HPC does not formalize such language at the moment, the language used must possess certain qualities such as sharing high level concepts with HPCML. Actually, the only language supported by the ArchiMDE tool is the textual DSL from the Arcane platform [8].

The management of constants holds an important role in scientific software. To fulfill this task HPCML extends the *numerical* metamodel from the Paprika tool [13] and provides constructs to define data set of constants (*HPCDataSet*) for each *HPCComponent*. *HPCDataSet* are shared within components and can be used by *HPCFunction* through *HPCConstData*. It is possible to obtain the global data set of an application by merging all the data set from the components used. In this merging process, identical constants from different data sets are mapped and editable constants are selected. The global data set model resulting from this process is then passed to the Paprika tool that will generate a GUI to manage instances of this model (creation, edition).

The concrete syntax of the concepts defined in the structure package is based on a master-detail interface. The master area is a tree displaying the hierarchy of components and their elements. The detail area is form-based.

3. BEHAVIORAL ASPECT

The diversity and complexity of parallel architectures is problematic for the development of portable and efficient simulation software. Usually, the architectures of supercomputers are classified according to their memory architecture (e.g. is memory shared or distributed between the different processing units ?) or to the way they process data (e.g. is the same operation applied to multiple data ?). Modern architectures are frequently built by combining basic types of parallel architectures (e.g. an architecture based on several big computational nodes with a distributed memory architecture where each big node possesses a shared memory architecture). Due to this diversity, developers have to choose a programming solution for each kind of architecture they target. The most popular solutions in the HPC community are the MPI standard [17] for dealing with distributed memory architectures, OpenMP [4] for shared memory systems and CUDA [10] for exploiting GPUs. These three solutions rely on different programming paradigms that were influenced by the architecture they target.

In these conditions it is thus challenging to provide an abstract way of expressing parallelism that would still be compatible with current and possibly future architectures. The behavioral package provides concepts to compose an application from components and attempts to provide generic constructs for expressing potential parallelism. The metamodel of this package is presented in Figure 2.

The root element enabling to compose components is the *HPCFlowDescriptor*. It can be seen as merge of activity diagram formalism from UML [1] and SysML [6] as it provides a way to specify both control flow (*HPCFlowNode*, *HPCTransition*) and data flow (*HPCDataPortConnector*, *HPC-*

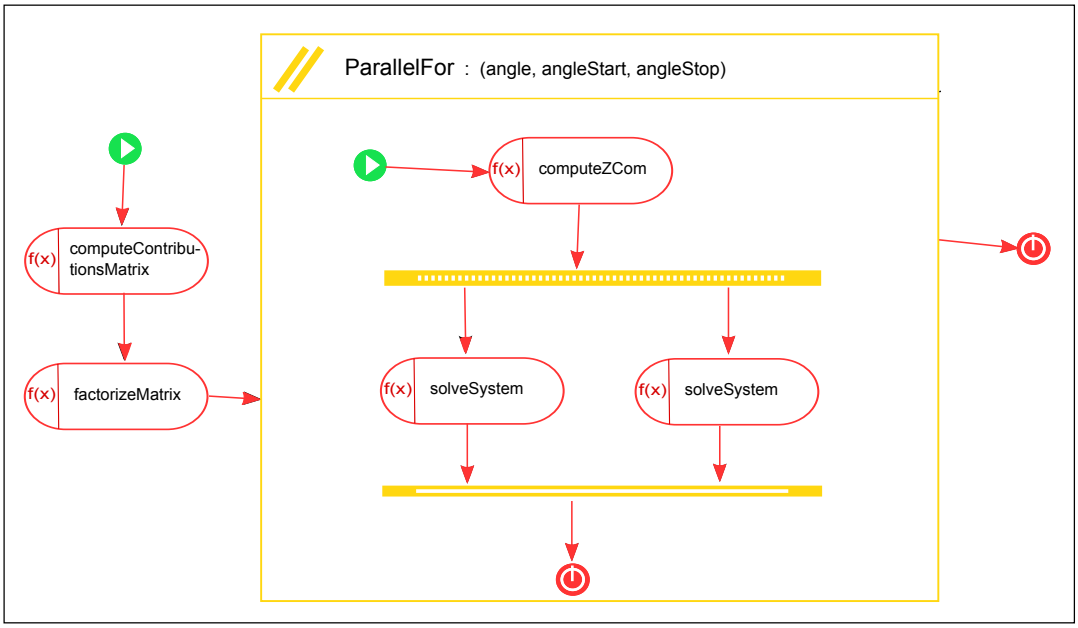

Figure 3: Example of the concrete syntax of an *HPCFlowDescriptor*

DataPort)

The current version of HPCML provides four ways of expressing parallelism in an application. Several of the constructs provided share the philosophy and were inspired by parallel patterns [11, 9] and algorithmic skeletons [3].

- Task parallelism can be specified in the control flow by using the concept of fork (*HPCFork*) in conjunction with the concept of join (*HPCJoin*).
- Data parallelism can be expressed with a domain decomposition approach (*HPCSetEnumerator*). The developer has just to specify which set may be partitioned (*HPCSet*). Additionally, the size of the ghost zone required at the boundaries of each partition may be specified to guide the decomposition process.
- Task parallelism can also be specified with loops that possess parallel semantics (*HPCParallelFor*). In this case the critical point is to determine dependencies among tasks. Three categories of dependence are identified (*HPCLoopDependenceCategory*): none (tasks have no dependence), accumulation into a shared data structure (this kind of dependency can be managed with the use of a *reduction* operation), complex dependencies.
- Parametric studies provides, at a meta level, a way of expressing task parallelism. A parametric study relies on the variation of parameters or functions to find the best compromise for a given requirement. Concepts for describing such parametric studies are not present in Figure 2 as they are defined in a separated package of HPCML.

Thanks to the level of abstraction of these concepts, it is possible to define transformations that could map them to different parallel architectures. For example an *HPCSetEnumerator* could be mapped onto a shared or distributed memory architecture.

The theory proposed by Moody in [12] was used for defining the concrete syntax of concepts from the behavior package. This theory aims at designing cognitively effective visual notations. Even though we cannot present entirely the concrete syntax and its design rationale in this paper, the figure 3 provides an overview. This figure shows the default view of an *HPCFlowDescriptor* : only the control flow is displayed. Several elements of the concrete syntax are inspired from existing notations (eg. *HPCFork* and *HPCJoin*) but were enhanced based on Moody's principles (increase of the perceptual discriminability and semantic transparency in the case of the *HPCFork* and *HPCJoin*). It is also interesting to note that the color is used to highlight the different concerns : orange for parallelism, red for control flow, blue for data flow (not shown here).

4. RELATED WORK

In addition to mainstream programming solutions such as the ones cited in section 3 (MPI [17], OpenMP [4] and Cuda [10]), research efforts to overcome at least one of the problems introduced in section 1 are numerous. Among them we can mention the High Productivity Computing Systems (HPCS) programme launched in 2002 by the DARPA [19] from which emerged novel programming languages: Chapel (Cray), Fortress(SUN) and X10 (IBM). The principal drawback of this approach is the need to develop high-performance compiler, debugger and implementation for each existing architecture. Macro-based approaches such as HMPP (Hybrid Multi-core Parallel Programming environment) [2] offer a respectable solution for improving legacy code. However, as their use is based on compiler directives which limit the separation of concerns, this solution can appear as less attractive for new developments.

The underlying idea to use model-driven engineering for developing scientific applications is also used in the Gaspard project [18, 16]. Although their work is step further in the right direction, we believe that their choice to model the application with the MARTE profile [7] is not optimal in term of accessibility for mathematicians and physicists.

Another project tackling our problematic by raising the level of abstraction is Liszt [5]. It aims at solving partial-differential equations on meshes by providing a domains-specific language built on top of Scala. Due to its qualities and common philosophy with HPCML, we plan to add Liszt as an additional target of the transformation process in the ArchiMDE tool. This would demonstrate the capability of HPCML to target multiple high-level platform (Arcane, Liszt). A more complete view of related work can be found in [15].

5. DISCUSSION

In this paper we addressed the main concepts of the HPCML metamodel and especially how parallelism can be expressed in a generic fashion. This metamodel and, more generally the MDE4HPC approach are currently used to redevelop an existing application used in production at CEA/CESTA. The goal of this experiment is to assess their ability to model and manage the different aspects of a real-scale simulation software.

Regarding future work, the definition of a textual language for describing algorithms (*HPCAlgorithmicContent*) is our next step. The syntax of the Liszt language is a good candidate and its implementation based on Scala makes it possible to use it in ArchiMDE as a generic language for the description of algorithms.

Another path of research concerns the transformation process. As a matter of fact HPCML provides only concepts for applied mathematics researchers to let them model how they want to solve their problems but what about the software engineers in charge of writing the model transformations that take the abstract description and produce the executable implementations. The actual situation is not idyllic as the transformations are quite "monolithic" and their parametrization limited. Hence they can be used for several projects but not for several platforms. We have already some ideas on how to model precisely the platform and on how to guide the mapping process but there is still a long way to go before getting a viable solution.

Until now, the use of modeling techniques in our context was aimed at easing development and maintenance processes. The models obtained in this process could be exploited in directions that are "classical" for the modeling community, but are not used these days by the HPC community. These extras, could include the formal verification of properties on the model, a better (and possibly formal) characterization of the needs for particular algorithms in terms of platform resources or architectures, an abstract description of timing properties that may lead to model level time analysis for existing algorithms, etc. The fact that there is no direct demand for such analysis by the HPC community is mainly due to the current habits of having low-level code that is not very exploitable by analysis tools. Adding more abstraction, would probably open the door to a wide range of model level analysis, most of them being yet to be defined in order to better address the needs of the HPC community.

6. REFERENCES

[1] UML 2.2 Superstructure Specification. Technical report, Object Management Group (OMG), 2009.

[2] F. Bodin. Keynote: Compilers in the manycore era. In *HiPEAC '09: Proceedings of the 4th International Conference on High Performance Embedded Architectures and Compilers*, pages 2–3, Berlin, Heidelberg, 2009. Springer-Verlag.

[3] M. Cole. *Algorithmic skeletons: structured management of parallel computation*. MIT Press, Cambridge, MA, USA, 1991.

[4] L. Dagum and R. Menon. Openmp: An industry-standard api for shared-memory programming. *Computing in Science and Engineering*, 5:46–55, 1998.

[5] Z. DeVito, N. Joubert, F. Palacios, S. Oakley, M. Medina, M. Barrientos, E. Elsen, F. Ham, A. Aiken, K. Duraisamy, E. Darve, J. Alonso, and P. Hanrahan. Liszt: a domain specific language for building portable mesh-based pde solvers. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 9:1–9:12. ACM, 2011.

[6] S. Friedenthal, A. Moore, and R. Steiner. *A Practical Guide to SysML: Systems Modeling Language*. Morgan Kaufmann Publishers Inc., 2008.

[7] S. Gérard and B. Selic. The uml marte standardized profile. In *Proceedings of the 17th IFAC World Congress*, pages 6–11, 2008.

[8] G. Grospellier and B. Lelandais. The arcane development framework. In *POOSC '09: Proceedings of the 8th workshop on Parallel/High-Performance Object-Oriented Scientific Computing*, pages 1–11, New York, NY, USA, 2009. ACM.

[9] K. Keutzer, B. L. Massingill, T. G. Mattson, and B. A. Sanders. A design pattern language for engineering (parallel) software: merging the plpp and opl projects. In *Proceedings of the 2010 Workshop on Parallel Programming Patterns*, ParaPLoP '10. ACM, 2010.

[10] D. Kirk. Nvidia cuda software and gpu parallel computing architecture. In *ISMM*, pages 103–104, 2007.

[11] T. Mattson, B. Sanders, and B. Massingill. *Patterns for parallel programming*. Addison-Wesley Professional, first edition, 2004.

[12] D. L. Moody. The "physics" of notations: Toward a scientific basis for constructing visual notations in software engineering. *IEEE Transactions on Software Engineering*, 35:756–779, 2009.

[13] D. Nassiet, Y. Livet, M. Palyart, and D. Lugato. Paprika: Rapid UI development of scientific dataset editors for high performance computing. In *SDL 2011: Integrating System and Software Modeling*, pages 69–78. Springer, 2011.

[14] M. Palyart, D. Lugato, I. Ober, and J. Bruel. Improving scalability and maintenance of software for high-performance scientific computing by combining MDE and frameworks. In *MODELS'11, Model Driven Engineering Languages and Systems*, pages 213–227.

Springer, 2011.

[15] M. Palyart, D. Lugato, I. Ober, and J.-M. Bruel. MDE4HPC: An approach for using Model-Driven Engineering in High-Performance Computing. In *15th System Design Languages Forum (SDL 2011)*, 2011.

[16] W. Rodrigues, F. Guyomarc'h, and J. Dekeyser. An mde approach for automatic code generation from uml/marte to opencl. *Computing in Science Engineering*, (99):1, 2012.

[17] M. Snir, S. W. Otto, D. W. Walker, J. Dongarra, and S. Huss-Lederman. *MPI: The Complete Reference*. MIT Press, Cambridge, MA, USA, 1995.

[18] J. Taillard, F. Guyomarc'h, and J.-L. Dekeyser. A Graphical Framework for High Performance Computing Using An MDE Approach. *Euromicro Conference on Parallel, Distributed, and Network-Based Processing*, 2008.

[19] M. Weiland. Chapel, Fortress and X10: Novel Languages for HPC. Technical report, The University of Edinburgh, October 2007. `http://www.hpcx.ac.uk/research/hpc/technical_reports/HPCxTR0706.pdf`.